Scientific Computing III. High Performance Scientific Computing

(Phys 2109/Ast 3100H)

Lecture 2: Parallel Programming Paradigms

SciNet HPC Consortium, University of Toronto

Winter 2013

SciNet compute • calcul CANADA

Parallel Computers

Top500.org:

List of the worlds 500 largest supercomputers. Updated every 6 months,

Info on architecture, etc.

Home ▸ Lists ▸ November 2010

TOP500 List - November 2010 (1-100)

R_{max} and R_{peak} values are in TFlops. For more details about other fields, check the TOP500 description.

Power data in KW for entire system

next

Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00	4701.00	4040.00
2	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.60
3	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00	2984.30	2580.00
4	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.61
5	DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00	1288.63	2910.00
6	Commissariat a l'Energie Atomique (CEA) France	Tera-100 - Bull bullx super-node S6010/S6030 / 2010 Bull SA	138368	1050.00	1254.55	4590.00

Supercomputer architectures

- Clusters, or, distributed memory machines
  In essence a bunch of desktops linked together by a network ("interconnect"). Easy and cheap.
- Multi-core machines, or, shared memory machines
  These can see the same memory. Limited number of cores, typically, and much more $$$.
- Vector machines.
  These were the early supercomputers, and could do the same operation on a large number of numbers at the same time.
  Very $$$$$$, especially at scale.
  These days, most chips have some low-level, small size vectorization, but you rarely need to worry about it (compiler should do this).

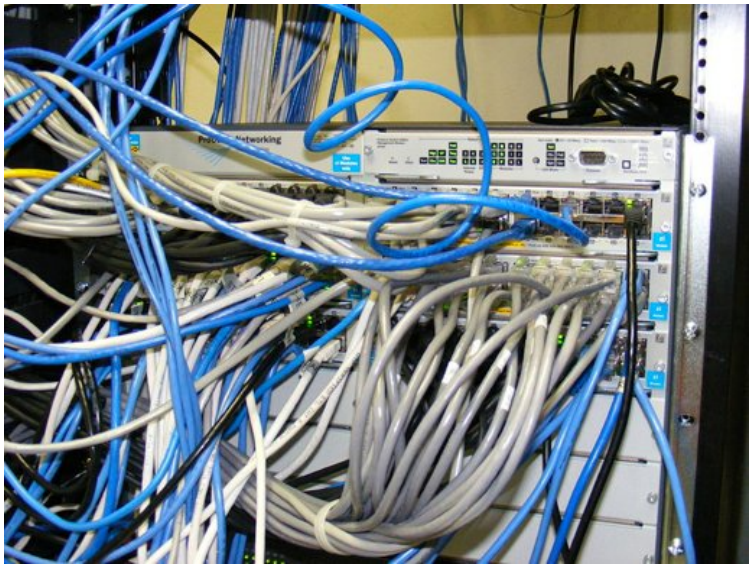
Most supercomputers are a hybrid combo of these different architectures.

SciNet
compute • calcul
CANADA

Distributed Memory: Clusters

Simplest type of parallel computer to build

- Take existing powerful standalone computers
- And network them



(source: http://flickr.com/photos/eurleif)

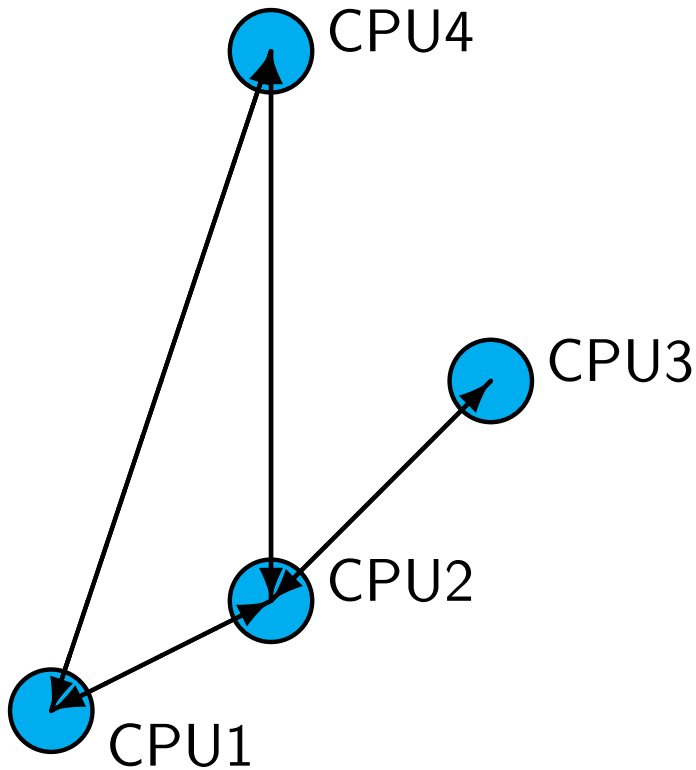
SciNet compute • calcul CANADA

Distributed Memory: Clusters

Each node is independent!

Parallel code consists of programs running on separate computers, communicating with each other.

Could be entirely different programs.



Distributed Memory: Clusters

Each node is independent!

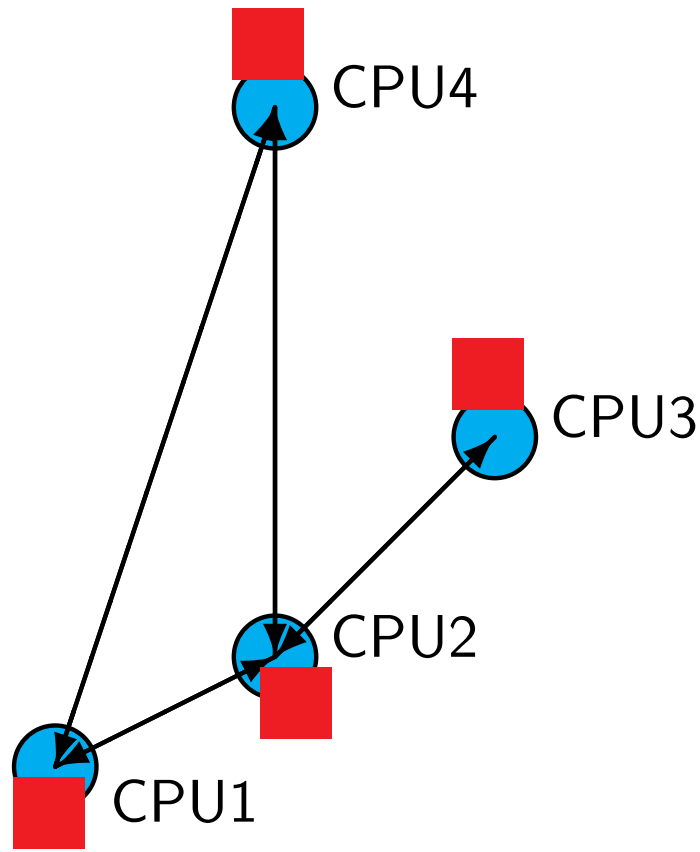
Parallel code consists of programs running on separate computers, communicating with each other.

Could be entirely different programs.

Each node has own memory!

Whenever it needs data from another region, requests it from that CPU.

Usual model: "message passing"



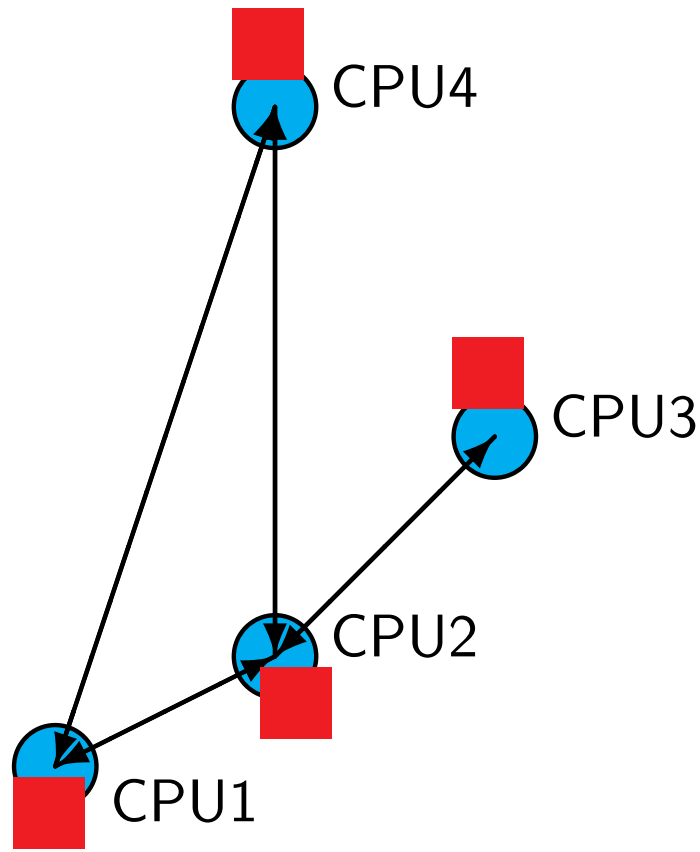
Clusters+Message Passing

Hardware:

Easy to build
(Harder to build well)
Can build larger and
larger clusters relatively
easily

Software:

Every communication
has to be hand-coded:
hard to program



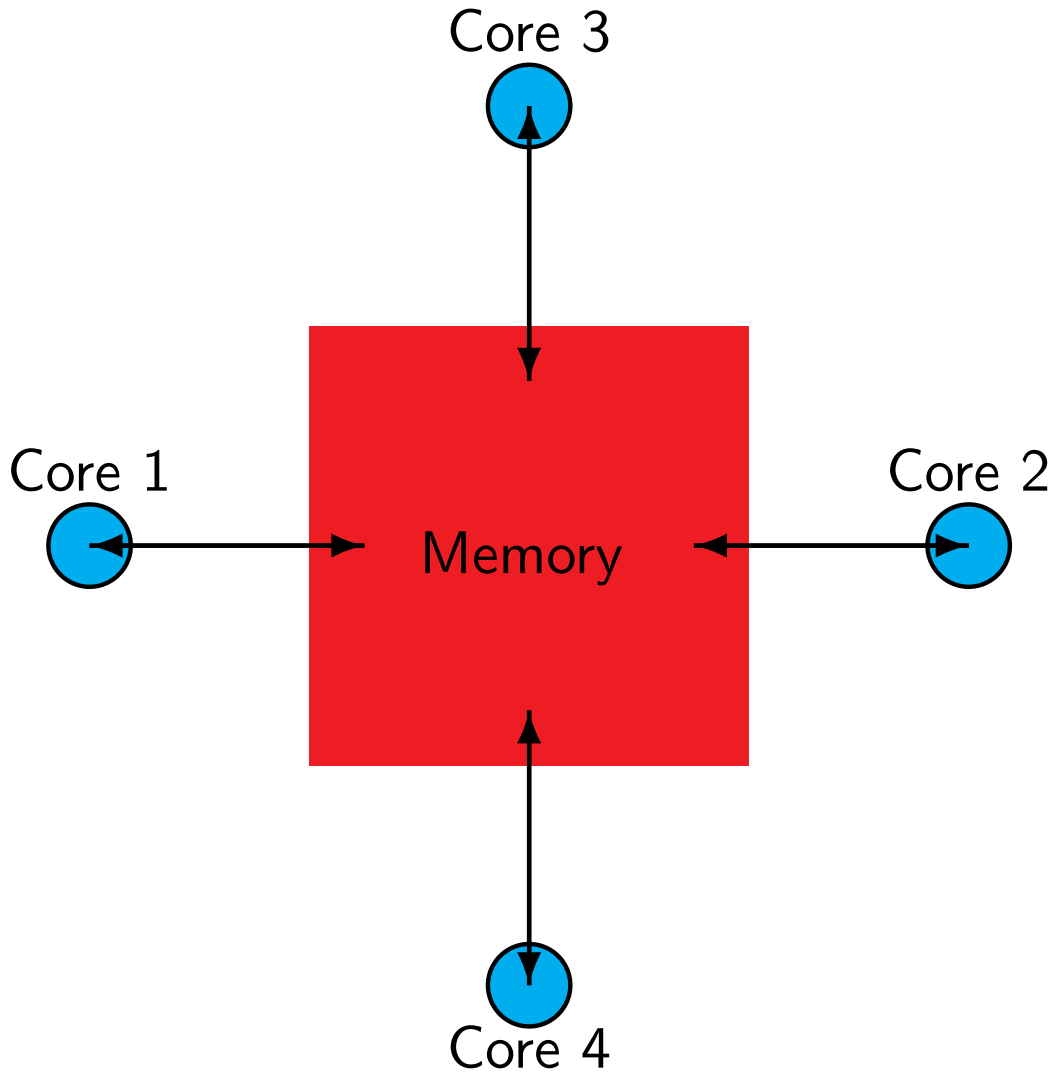
Shared Memory

One large bank of memory, different computing cores acting on it. All 'see' same data.

Any coordination done through memory

Could use message passing, but no need.

Each code is assigned a thread of execution of a single program that acts on the data.



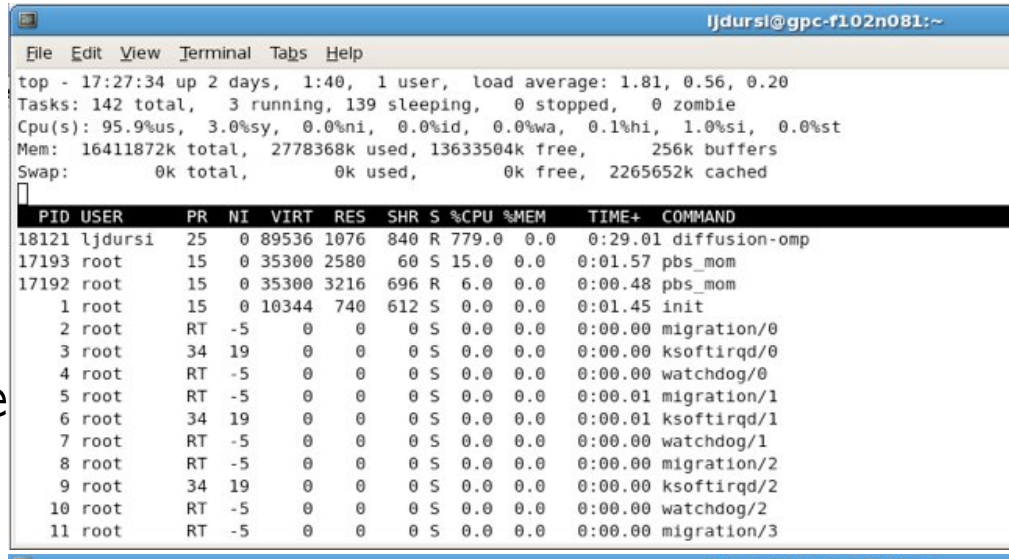
Threads versus Processes

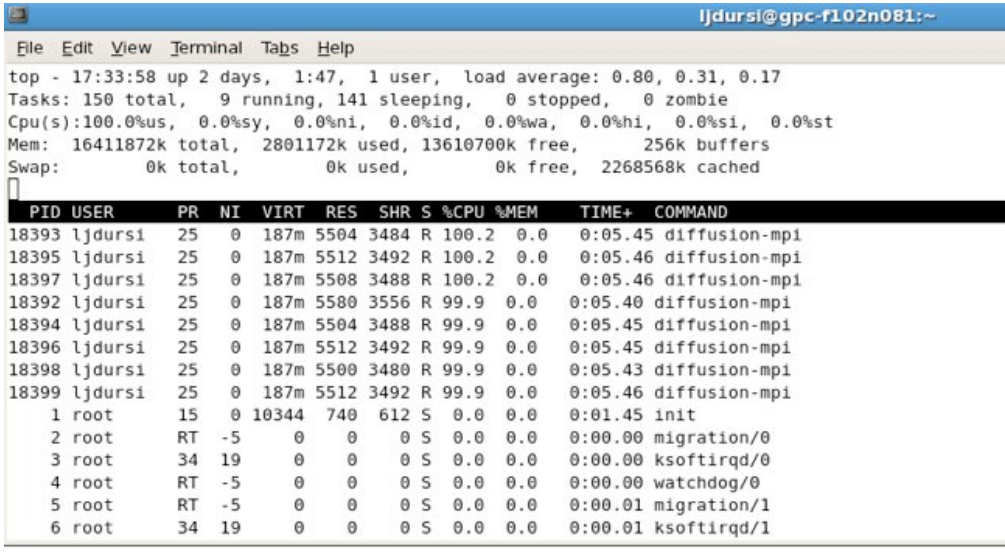
Threads:

Threads of execution within one process, with access to the same memory etc.

Processes:

Independent tasks with their own memory and resources



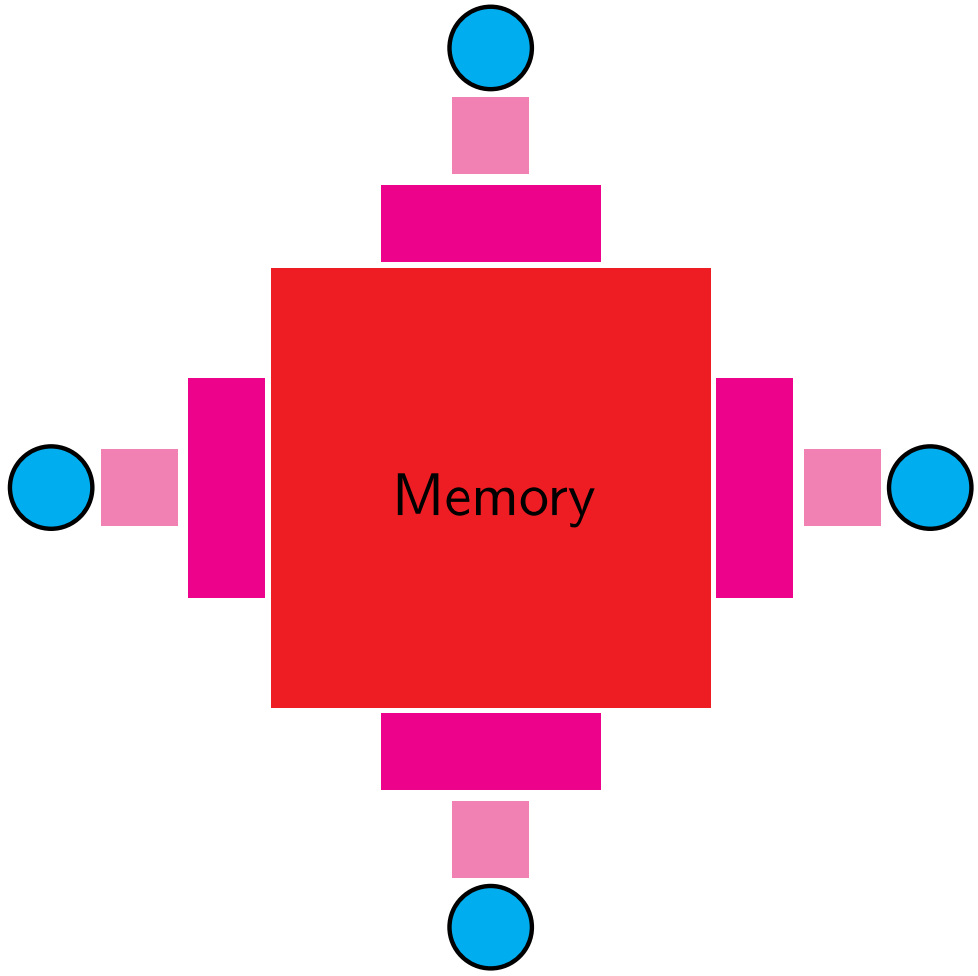


SciNet
compute • calcul
CANADA

Shared Memory: NUMA

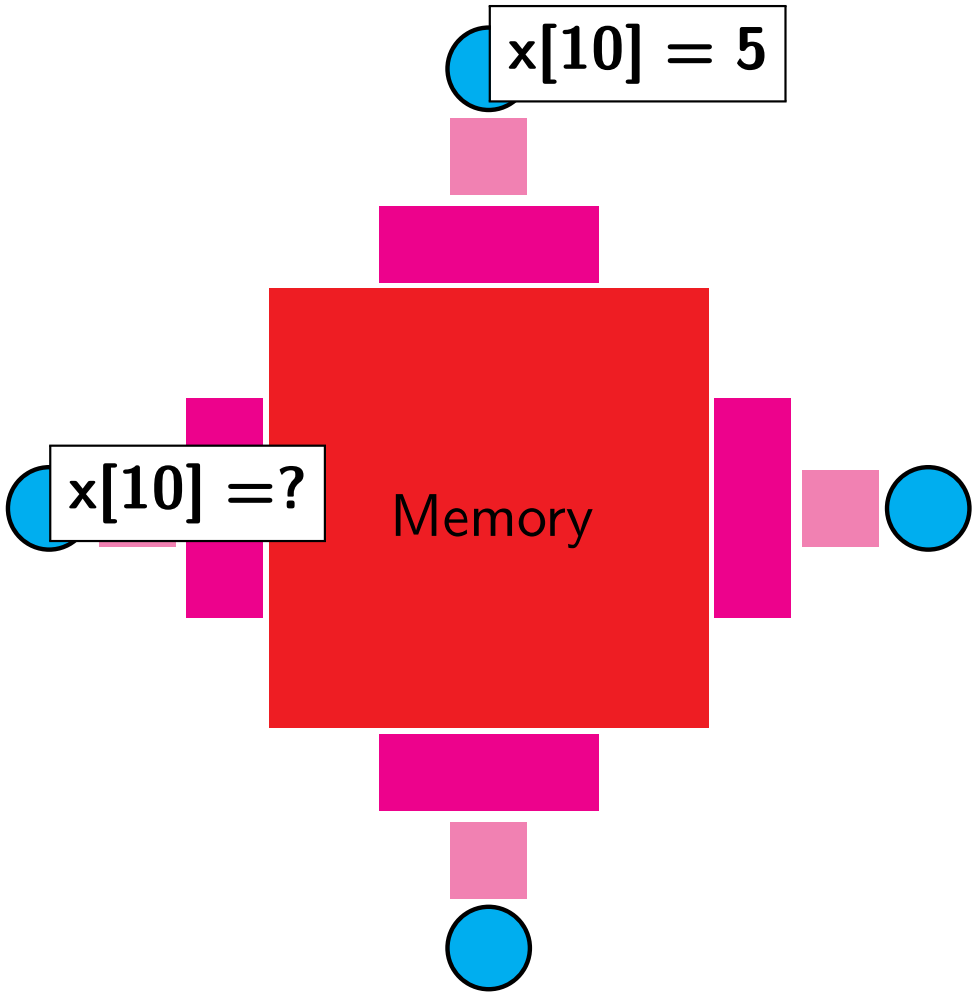
Non-Uniform Memory Access

- Each core typically has some memory of its own.
- Cores have cache too.
- Keeping this memory coherent is extremely challenging.



Coherency

- The different levels of memory imply multiple copies of some regions
- Multiple cores mean can update unpredictably
- Very expensive hardware
- Hard to scale up to lots of processors, very $$$
- Very simple to program!!



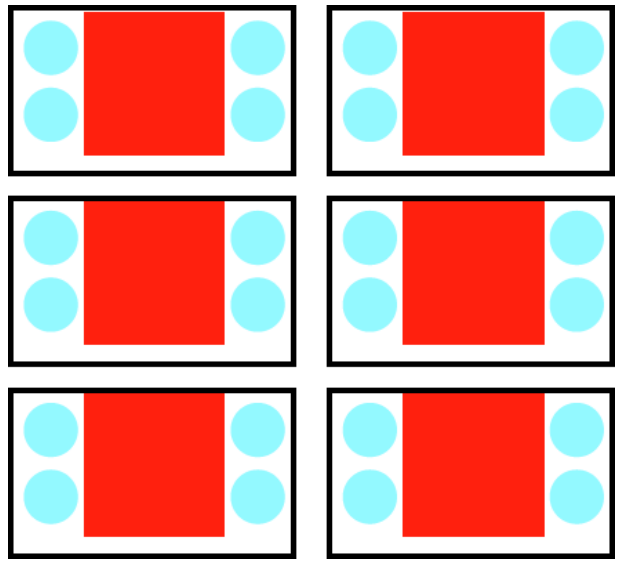
Shared Memory Communication Cost

	Latency	Bandwidth
GigE	10 μs (10,000 ns)	1 Gb/s (60 ns/double)
Infiniband	2 μs (2,000 ns)	2-10 Gb/s (10 ns /double)
NUMA (shared memory)	0.1 μs (100 ns)	10-20 Gb/s (4 ns /double)

Processor speed: O(GFLOP) $\sim$ few ns or less.

Hybrid Architectures

- Multicore machines linked together with an interconnect
- Many cores have modest vector capabilities.
- Machines with GPU or other coprocessors: GPU is multi-core, but the amount of shared memory is limited.



Techniques for different parallel programming paradigms

Techniques for different parallel programming paradigms

- Embassingly parallel applications
  Scripting, GNU Parallel[1]

SciNet compute • calcul CANADA

Techniques for different parallel programming paradigms

- Embassingly parallel applications
  Scripting, GNU Parallel[1]
- Shared memory
  OpenMP, Threads, Automated parallelization

SciNet
compute • calcul
CANADA

Techniques for different parallel programming paradigms

- Embassingly parallel applications
  Scripting, GNU Parallel[1]
- Shared memory
  OpenMP, Threads, Automated parallelization
- Distributed memory
  MPI, Files

Techniques for different parallel programming paradigms

- Embassingly parallel applications
  Scripting, GNU Parallel[1]
- Shared memory
  OpenMP, Threads, Automated parallelization
- Distributed memory
  MPI, Files
- Graphics computing
  CUDA, OpenCL

Techniques for different parallel programming paradigms

- Embassingly parallel applications
  Scripting, GNU Parallel[1]
- Shared memory
  OpenMP, Threads, Automated parallelization
- Distributed memory
  MPI, Files
- Graphics computing
  CUDA, OpenCL
- Hybrid combinations

[1] O. Tange (2011): *GNU Parallel - The Command-Line Power Tool*, ;login: The USENIX Magazine, February 2011:42-47.

SciNet compute • calcul CANADA

Data or computation bound?

Computation bound $\Rightarrow$

Task parallelism

Focus on processes/threads.
These processes may have quite different computations to do.
Bring the data to the computation

Data bound $\Rightarrow$

Data parallelism

Focus on operations on a given, large data set. Data set often an array, partitioned, and tasks act on separate partitions.
Bring the computation to the data

Granularity

Fine-grained parallelism

Small individual tasks.
The data is transferred among processors frequently.
Often OpenMP, to overlap different hardware functions

Coarse-grained parallelism

Data communicated infrequently,
after larger amounts of computation.
Often MPI, because of network latency

Too fine-grained $\Rightarrow$ overhead
Too course-grained $\Rightarrow$ load imbalance

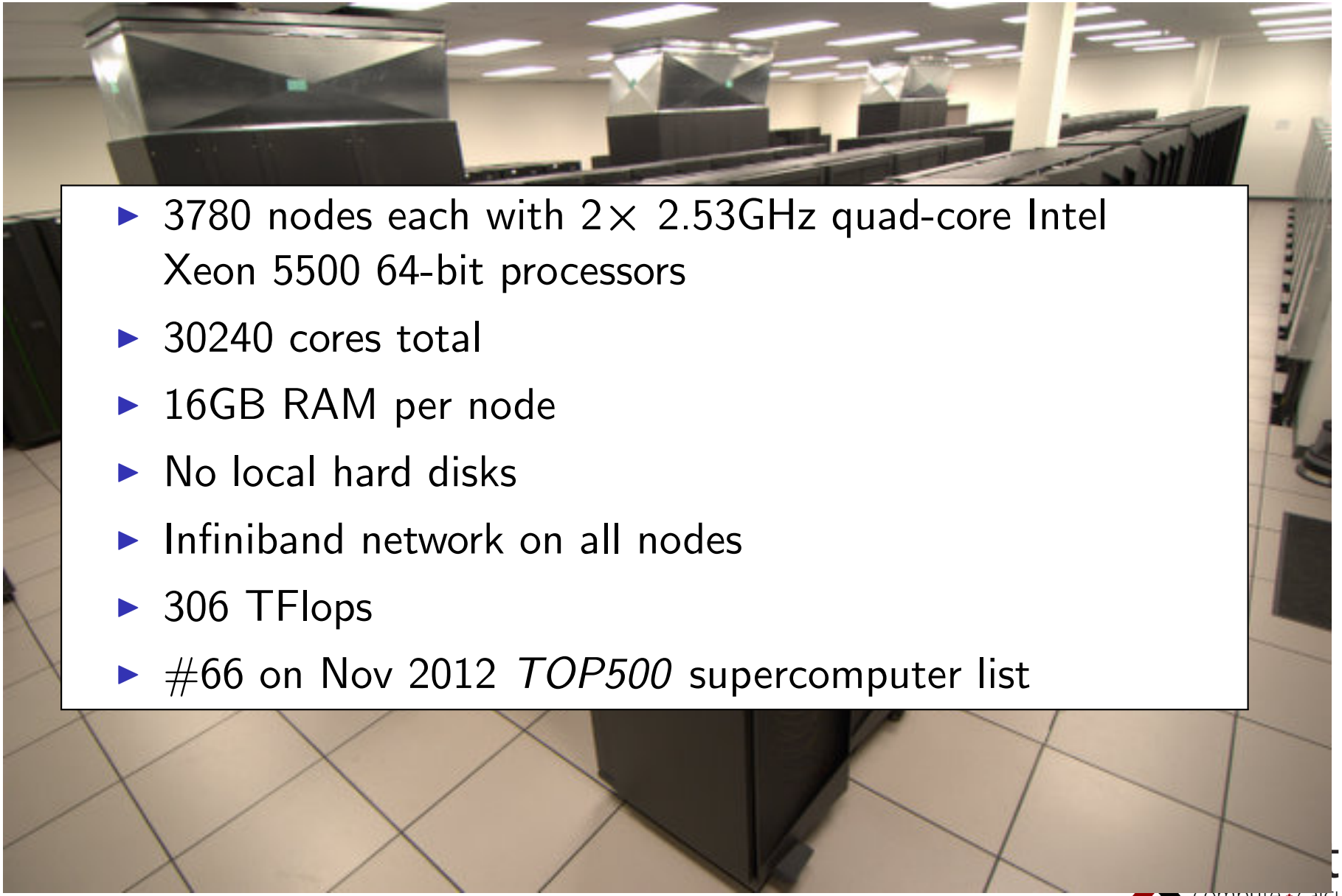
Balance depends on architecture, access pattern, and computation.

Using SciNet

GPC

Using SciNet

GPC



- 3780 nodes each with 2× 2.53GHz quad-core Intel Xeon 5500 64-bit processors
- 30240 cores total
- 16GB RAM per node
- No local hard disks
- Infiniband network on all nodes
- 306 TFlops
- #66 on Nov 2012 *TOP500* supercomputer list

Mini intro to SciNet - how to get started

- Need to have an account
- If you don't: get it (wiki.scinethpc.ca/wiki/index.php/Essentials)
- If you can't: email us.
- Read the SciNet Tutorial and the GPC quick start on the wiki. (wiki.scinethpc.ca/wiki/index.php/GPC_Quickstart)

Access:

```
$ ssh -X login.scinet.utoronto.ca
$ ssh -X gpc01 (or gpc02, gpc03, gpc04)
```

SciNet compute • calcul CANADA

Mini intro to SciNet - how to compile

- Ssh into one of the devel nodes gpc0{1,2,3,4}.
- Then load a compiler into your environment:

```
$ module load gcc
```

- Then you can compile using g++.
- In general, the Intel compilers are preferred on the GPC, but for the purpose of this course, g++ is fine too.

SciNet compute • calcul CANADA

Mini intro to SciNet - how to run

- You do not run on the devel nodes.
- You run on compute nodes.
- Compute nodes are reserved through the queue.
- You can get an interactive session on a compute nodeby making the following request to the scheduler (from one of the devel nodes)

```
$ qsub -l nodes=1:ppn=8,walltime=2:00:00 -I -X -qdebug
```

  which gets a dedicated compute node for two hours. You can ask for more time but you will need to omit the '-qdebug' and you may be waiting for a long time for a node.
- Alternatively, submit a job script.

SciNet compute • calcul CANADA

GNU Parallel

- What if you need to keep all 8 cores on a node busy.
- GNU Parallel can help you with that!
- GNU parallel is a really nice tool to run multiple serial jobs in parallel. It allows you to keep the processors on each 8core node busy, if you provide enough jobs to do.
- GNU parallel is accessible on the GPC in the module gnu-parallel.

```
$ module load gnu-parallel/20120622
```

Note that we recommend the newer version of gnu-parallel over the (default) 2010 one.

GNU Parallel Example

```
gpc-f101n084-$ g++ -O3 mycode.cc -o mycode
gpc-f101n084-$ mkdir $SCRATCH/example2
gpc-f101n084-$ cp mycode $SCRATCH/example2
gpc-f101n084-$ cd $SCRATCH/example2
gpc-f101n084-$ cat > joblist.txt
  mkdir run1; cd run1; ../mycode 1 > out
  mkdir run2; cd run2; ../mycode 2 > out
  ...
  mkdir run64; cd run64; ../mycode 64 > out
gpc-f101n084-$ cat > myjob.pbs
    #!/bin/bash
    #PBS -l nodes=1:ppn=8,walltime=24:00:00
    #PBS -N GPJob
    cd $PBS_O_WORKDIR
    module load gcc gnu-parallel/20120622
    parallel -j 8 < joblist.txt
gpc-f101n084-$ qsub myjob.pbs
    2961985.gpc-sched
gpc-f101n084-$ ls
    GPJob.e2961985  GPJob.o2961985  joblist.txt
    mycode          myjob.pbs/      run3/
    ...
```

SciNet compute • calcul CANADA

GPC and hyperthreading

- HyperThreading: Appears as if there are 16 processors rather than 8 per node.
  - For OpenMP applications this is the default unless `OMP_NUM_THREADS` is set.
  - For MPI, try `-np 16`.
  - For gnu parallel, use `-j 16`.
- Always request ppn=8, even with hyperthreading.
- *Always test if this is beneficial and feasible!*

Homework

- Make sure you've got a SciNet account!
- Read the SciNet tutorial (as it pertains to the GPC)
- Read the GPC Quick Start.
- Get the first set of code:

```
$ cd $SCRATCH
$ git clone /scinet/course/sc3/homework1
$ cd homework1
$ source setup
$ make
```

- Contains threaded program 'blurppm' and 266 ppm images to be blurred.
- Usage:

```
blurppm INPUTPPM OUTPUTPPM BLURRADIUS NUMBEROFTHREADS
```

Homework

- Simple test:

```
$ qsub -l nodes=1:ppn=8,walltime=2:00:00 -I -X -qdebug
$ cd $SCRATCH/homework1
$ blurppm 001.ppm new001.ppm 30 1
$ display 001.ppm &
$ display new001.ppm &
```

- Will want to do timing tests:

```
$ time blurppm 001.ppm new001.ppm 30 1
real 0m52.900s
user 0m52.881s
sys 0m0.008s
```

Homework

Part 1

- Time blurppm with a BLURRADIUS ranging from 1 to 41 in steps of 4, and for NUMBEROFTHREADS ranging from 1 to 16. Record the (real) duration of each run.
- Plot the duration as a function of NUMBEROFTHREADS, as well as the speed-up and efficiency.
- Submit script and plots of the duration, speedup and effiency as a function of NUMBEROFTHREADS.

Homework

Part 2

- Use GNU parallel to run blurppm on all 266 images with a radius of 41.
- Investigate different scenarios:
  1. Have GNU parallel run 16 at a time with just 1 thread.
  2. Have GNU parallel run 8 at a time with 2 threads.
  3. Have GNU parallel run 4 at a time with 4 threads.
  4. Have GNU parallel run 2 at a time with 8 threads.
  5. Have GNU parallel run 1 at a time with 16 threads.

  Record the total time it takes in each of these scenarios.
- Repeat this with a BLURRADIUS of 3.
- Submit scripts, timing data and plots.